Max-sum for allocation of changing cost tasks

James Parker[1], Alessandro Farinelli[2], and Maria Gini[1]

[1] Computer Science and Engineering, University of Minnesota
jparker@cs.umn.edu and gini@cs.umn.edu
[2] Department of Computer Science, University of Verona
alessandro.farinelli@univr.it

Abstract. We present a novel decentralized approach to allocate agents to tasks whose costs increase over time . Our model accounts for both the natural growth of the tasks and the effort of the agents at containing such growth. The objective is to minimize the increase in task costs. We show how a distributed coordination algorithm, which is based on max-sum, can be formulated to include costs of tasks that grow over time. Considering growing costs enables our approach to solve a wider range of problems than existing methods. We compare our approach against state-of-the-art methods in both a simple simulation and RoboCup Rescue simulation.

1 Introduction

Task allocation problems are ubiquitous. We address task allocation for problems where the costs for completing the tasks increase over time. Applications for this type of problem include minimizing damage from invasive species, resource distribution for fighting epidemics, and containment of wild fires. In most practical applications, multiple agents need to cooperate to complete all the tasks. (i.e., fire fighters cooperate to extinguish large fires). Since costs grow over time, a task can grow indefinitely if not enough agents are assigned to it. If the growth rate surpasses the reduction the agents can provide, then that task cannot be completed.

A formulation for this problem and a centralized heuristic method was proposed in [8]. We use the same formulation of the cost growth model and propose the first decentralized solution for this class of task allocation problems. Our method is based on max-sum [2], which we modify by projecting future growth into our formulation. We assume that an approximation of the growth functions is known upfront, but they could be learned. New tasks of any size can appear at any time and agents may disappear.

This paper provides the following main contributions: (i) a general approach to handle growth functions for problems where the costs of tasks increase over time; (ii) an empirical evaluation and comparison with state-of-the-art methods both in a simple simulation and in RoboCup Rescue [3]. The results show our decentralized method performs similarly to the state-of-the-art centralized method.

2 Related Work

Multi-agent task allocation with temporal constraints has been studied from a variety of angles. Efficient auction based methods for task with disjoint time windows are presented in [6]. Their method is not be directly applicable to our problem because in our case the cost of the tasks changes over time and hence the rewards given to the agents for completing tasks have to depend on the actions of other agents. The work in [16] focuses on tasks that can be completed only with multiple agents but assumes task costs are fixed. In [1] Fisher market clearing with soft deadlines is used to assign agents to tasks when reward for task completion decreases over time. In their work tasks become less important over time, while in our work tasks become harder over time and there can be severe consequences if tasks are not completed (i.e., the city might burn completely).

A decentralized solution using max-sum to allocate tasks in RoboCup Rescue is presented in [14]. The work focuses on coalition formation among agents taking into account both temporal and spatial constraints. Their objective is to complete as many tasks as possible when each task has a deadline for being completed. In their case tasks have a fixed cost, while we assume costs grow over time. Their experimental results are obtained using synthetic data not in RoboCup Rescue. Other distributed approaches based on max-sum have been proposed for RoboCup Rescue, in particular using Tractable Higher Order Potentials (THOPs) [15] to improve the efficiency [11, 13]. The use of THOPs reduces the message passing of max-sum from exponential to polynomial by exploiting the structure of the problem. To gain these benefits, problems must be formulated with binary variables and with more limited types of factors. The closest to our work is the work in [12], which uses binary max-sum. They evaluate the utility of the current state using a domain specific heuristic based on distance, intensity of fire, and number of agents assigned to a task. Our approach is more general because it is based on a task cost growth model which is independent of the specific application domain. The incorporation of temporal reasoning on task growth, estimation of solution cost and generality across domains makes our work significantly different.

3 Problem Definition

We assume homogeneous agents which have a spatial location and tasks which have a location and have a cost that changes over time. An agent must be at a task's location in order to work on that task. We denote the set of homogeneous agents by $A = \{a_1, \ldots, a_n\}$, where $n = |A|$ and the set of tasks by $B = \{b_1, \ldots, b_m\}$, where $m = |B|$. The set of active agent assignments is denoted by $N^t = \{n_1^t, \ldots, n_{|B|}^t\}$, where n_i^t is the set of agents from A that are currently working on task b_i at time t. An agent can only work on one task at a time, so $n_i^t \subseteq A$ and $\forall i \neq j, n_i^t \cap n_j^t = \emptyset$. The travel time, $TT(x, y)$, between two locations, x and y, is assumed to be computable. Travel time causes a delay between the time an agent works on one task and the time it can work on another. This means $\sum_i n_i^t \leq |A|$ as some agents might be in transit.

Each agent provides work amount w per time unit once the agent has reached a task. Every task $b_i \in B$ has a cost defined with the following recursive relationship:

$$f_i^{t+1} = f_i^t + h_i(f_i^t) - w \cdot |n_i^t|, \tag{1}$$

where f_i^t starts at some initial cost f_i^0 and h_i is a positive semidefinite monotonically increasing function, as in [8]. The growth function h_i is assumed to be known but it could be estimated. Here we treat f_i^t as a sequence due to the use of discrete time steps, but it could be treated as a continuous function if that is a better model for the domain.

If at time t the cost of task b_i, namely f_i^t, reaches or goes below zero the task is considered complete. We denote this completion time as ct_i, For this reason, when f_i^t is non-positive, $h_i(f_i^t)$ is assumed to be zero and we do not allow those tasks to be assigned to agents. The time when the last task is completed is defined as t_s, namely $\max_{b_i \in B} ct_i$. Our objective is to minimize the accumulated growth of all tasks, denoted by

$$R_{t_s} = \sum_{t < \max_{b_i \in B} ct_i} \sum_{b_i \in B} h(f_i^t) \tag{2}$$

This objective corresponds to minimizing the amount costs increase over all time. For instance, in an epidemic outbreak this would mean allocating a limited number of doctors (agents) to areas with sick people (tasks). The sickness is contagious so areas with the disease have a higher infection rate, based on the growth rate $h_i(x)$. Our model projects the infection rate and the rate of doctors curing to decide where doctors should be located and when and if they need to move. The people who are sick before the crisis is discovered do not affect our goal, but we want to minimize R_{t_s}, which is the number of new people who become sick before the epidemic is cured.

4 Max-Sum Formulation

A Distributed Constraint Optimization problem (DCOP) can be formally defined as a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{x_1, x_2, ...x_{|\mathcal{X}|}\}$ is a set variables with domains $\mathcal{D} = \{D_1, D_2, ...D_{|\mathcal{X}|}\}$ and $\mathcal{C} = \{c_1, c_2, ...c_{|\mathcal{C}|}\}$ are utility functions. Each utility function, c_i, has a value based on a subset of $\mathcal{X}$ variables, $\{x_{i_1}, x_{i_2}, ...\}$, specifically $c_i : D_{i_1} \times D_{i_2} \times ... \rightarrow \Re$. We will denote the specific variables used for each utility function, c_i, by X_{c_i}. The max-sum algorithm finds an approximate solution to the sum of each utility function c_i, namely:

$$maximize \sum_{c_i \in C} c_i(X_{c_i})$$

Max-sum finds a solution by alternating between passing messages from variables to functions and from functions to variables. Both types of messages are repeatedly sent until they cease to change or a maximum number of iterations has been reached. After convergence, each variable can determine the best value by examining the most recent messages passed to it. Both message types generate outgoing messages based on all the other incoming messages.

A message from a variable to function simply adds all the messages coming into the variable from all the other functions. A message from a variable x_i to function c_j would be generated only if $x_i \in X_{c_i}$, namely this function uses the variable. The message generated is:

$$\mu_{x_i \rightarrow c_j}(d) = \sum_{k \in C_{x_i}/c_j} \mu_{k \rightarrow x_i}(d),$$

where C_{x_i} is the set of all functions which involve x_i and $d \in D_i$ is a value in the domain of x_i. $\mu_{k \to x_i}$ is the most recent message x_i received from function k.

The message from a function to variable is then:

$$\mu_{c_j \to x_i}(d) = \max_{Y} \left(c_j(d, Y) + \sum_{k \in X_{c_j}/x_i} \mu_{k \to c_j}(y_i) \right),$$

where $y_i \in D_i$ is a value of variable x_i, and Y is a set of values for variables X_{c_j}/x_i. Once messages have converged, each variable x_i chooses the value that maximizes all of its incoming messages, namely $\text{argmax}_d \sum_{k \in C_{x_i}} \mu_{k \to x_i}(d)$.

Approaches to solve DCOPs range from optimal techniques [7] to heuristics [2]. Optimal solutions require an exponential coordination overhead (i.e. communication computation). We approximate an unknown process which involves a potentially large number of agents, hence we use heuristic methods, which do not guarantee a solution quality but have a lower coordination overhead. Optimal methods would only solve the approximate model without guaranteeing an actual optimal solution. Following [12] we use binary max-sum in conjunction with Tractable Higher Order Potentials to more efficiently pass messages [15]. This can reduce the message passing complexity from $O(|B|^{|A|})$ to $O(|A| \log |A|)$. The use of binary variables means $D_i \in \{0, 1\}$.

The heuristics to prioritize tasks used in [12] are domain specific to RoboCup Rescue. Explicit values are assigned to fires based on their intensity and the number of agents assigned. A distance penalty discourages agents from selecting tasks far away. This is balanced with a utility heuristic, which discourages many agents from working on the same task. Our max-sum model instead assumes task cost size follows the recursive relationship in Eq. 1 and computes the expected effect of agent allocation on the tasks. We also compute a utility and distance penalty, but our distance penalty computes how much a task will grow before an agent reaches it, and our utility computes how much faster a task will be completed if an agent is assigned to it. This allows our max-sum formulation to generalize to any task allocation problem, as long as their cost can be approximated by the general recursive relationship in Eq. 1, unlike the work in [12]. Our method also solves a temporal component which is not addressed in [12].

Next we discuss the specifics of our max-sum problem formulation. We treat the cost, f_i^t, as an algebraic variable that depends on time t and the number of agents active $|n_i^t|$. The goal is to minimize R_{t_s}, the total amount of growth before all tasks are completed. Our task growth approximation for the model in Eq. 1 is $h(f_i^t) = g_i \cdot f_i^t$, where g_i is a constant found empirically. For ease of calculation, we assume that time is continuous rather than discrete so this means $\frac{\delta f_i^t}{\delta t} = g_i \cdot f_i^t$. Solving for f_i^t yields the well known exponential function: $f_i^t = K_i \cdot e^{g_i \cdot t}$, where K_i is the integration constant. The initial task cost, f_i^0, is used to determine an appropriate K_i for each task. The cost reduction from agents working on a task is represented as $\frac{\delta f_i^t}{\delta t} = g_i \cdot f_i^t - |n_i^t| \cdot w$, which can again be solved for f_i^t:

$$f_i^t = \frac{|n_i^t| \cdot w}{g_i} + K_i \cdot e^{g_i \cdot t}, \tag{3}$$

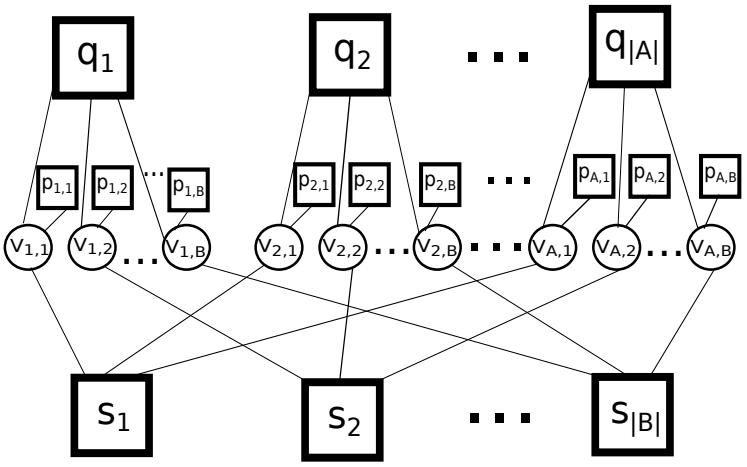


Fig. 1: The factor graph for our representation. v are binary indicators of an agent's assignment, p distance penalties, q constraints for ensuring an agent is active on one task, and s the utility function for the assignment (Eq. 5).

where again K_i is an integration constant. K_i is negative if the agents are completing the task faster than it is growing, namely $K_i < 0$ if and only if $|n_i^t| \cdot w > g_i \cdot f_i^t$. If the number of agents changes, K_i has to be recomputed. We can use further algebra to get the following equations:

$$f_i^t = \frac{|n_i^t| \cdot w}{g_i} + \left(f_i^0 - \frac{|n_i^t| \cdot w}{g_i} \right) \cdot e^{g_i \cdot t} \tag{4}$$

$$\begin{aligned} -L(|n_i^t|) &= \tfrac{|n_i^t| \cdot w}{g_i} \cdot (\ln(\alpha) + 1 - \alpha) + f_i^t \cdot (\alpha - 1), \\ \text{where } \alpha &= \tfrac{|n_i^t| \cdot w}{|n_i^t| \cdot w - f_i^t \cdot g_i} \end{aligned} \tag{5}$$

Eq. 4 estimates the cost of task b_i at time t when $|n_i^t|$ agents are assigned to it. Eq. 5 is the natural growth of task cost from time t until task i is finished. We use a negative sign since max-sum maximizes but we want to minimize. This means $\sum_i -L(|n_i^t|) = -(R_{t_s} - R_t)$, where R_{t_s} is from Eq. 2.

The factor graph [5] used to model this problem is shown in Figure 1, with circles for the variables and squares for the functions. Each agent a_i controls all $|B|$ binary task assignment indicator variables $v_{i,j}$, where j indicates the agent's assigned task, b_j. The constraint q_i ensures an agent is assigned to exactly one task, specifically:

$$q_i(v_{i,1}, v_{i,2}, ...v_{i,|B|}) = \begin{cases} 0 & \sum_j v_{i,j} = 1 \\ -\infty & \text{otherwise} \end{cases}. \tag{6}$$

s_j is the accumulated cost growth until the task is finished. Since we want to minimize the accumulated growth, the negative of this value is taken:

$$s_j(v_{1,j}, v_{2,j}, ...v_{|A|,j}) = -L(\sum_i v_{i,j}), \text{ with L as in Eq. 5.} \quad (7)$$

$p_{i,j}(v_{i,j})$ is a travel penalty, since s_j assumes the agent arrives instantly, the correct amount is subtracted if agent a_i is still traveling to task b_j. $TT(v_{i,j}, b_j)$ denotes the travel time between agent a_i and task b_j. Knowledge of $\hat{T}$, the time it takes the currently assigned agents to complete the task, is approximated by the previous assignment to compute $p_{i,j}(v_{i,j})$ as follows:

$$p_{i,j}(v_{i,j}) = \begin{cases} \frac{w \cdot e^{g_i \cdot \hat{T}}}{g_i} \cdot \left(e^{-g_i \cdot TT(v_{i,j}, b_j)} - 1\right) & v_{i,j} = 1 \\ 0 & v_{i,j} = 0 \end{cases} \quad (8)$$

This factor graph is binary and composed of only Tractable Higher Order Potentials, specifically only the cardinality of the inputs is required for s_j (number of agents assigned to the task) and q_i (number of tasks assigned to an agent) [15].

Our global utility function is a sum over all functions:

$$\sum_i q_i(v_{i,1}, v_{i,2}, ...) + \sum_i \sum_j p_{i,j}(v_{i,j}) + \sum_j s_j(v_{1,j}, v_{2,j}...)$$

This factor graph is cyclical, so max-sum is not guaranteed to find a global optimum. Our max-sum use the same framework as [12], based on the work of [17] for an efficient anytime solution. Max-sum is run at every time step for multiple reasons. New tasks may appear, new agents may join and current agents may leave. Also as both the modeling and solution are approximate, readjusting projections with the real outcomes improves accuracy. This solutions is also decentralized, compared with past work [8], which proposed a centralized solution.

5 Results

We compare the performance of max-sum against a modified version of RT-LFF [8]. The original RT-LFF tried to minimize the time the latest task finished. We changed RT-LFF to instead minimize the accumulated task cost so it is consistent with our current problem formulation. RT-LFF uses LFF [8] for this initial assignment, which is optimal when agents cannot be reassigned. RT-LFF can change assignments if a large gain is detected. This modified RT-LFF has a worst-case running time of $O(|A||B|^2)$, but is typically significantly less in practice compared to $O(|B||A| \log |A|)$ of binary max-sum.

The comparisons are done in both a controlled simple simulator and in the RMAS-Bench [4] extension of RoboCup Rescue to show the applicability to complex environments and to make our results easily comparable.

5.1 Simple Simulator

The simple simulator gives us greater analytical power than RoboCup Rescue since every aspect can be controlled. Here we use the same h growth function for all the tasks

Table 1: Accumulated growth cost, R_{t_s} for different setups in the simple simulator

$h(x)$	Initial costs	Optimal	Max-sum	RT-LFF	AllOnOne	Uniform
$0.000016 \cdot x^3$	{20, 15}	8.1333	8.1334	8.3724	15.852	12.121
$0.00019 \cdot x^2$	{25, 20, 10}	22.761	22.761	23.632	39.268	49.076
$0.0036 \cdot x$	{50, 30}	188.72	188.72	188.72	188.76	189.41
$0.02 \cdot \sqrt{x}$	{5, 10, 15, 20}	43.398	43.398	144.60	69.117	79.878
$0.02 \cdot \ln(x+1)$	{40, 30}	28.158	28.158	41.661	31.409	39.134
mixed	{25, 10, 10}		24.485	29.064	28.110	40.948
sigmoid	{40, 30}		41.218	100.99	26.127	92.530

except in the mixed experiments (see Table 1). We explicitly define the initial costs, f_i^0, growth functions, $h_i(x)$, and the effect of agents on the task, w, for any number of tasks.

We consider the optimal, max-sum, and RT-LFF. For additional comparison, we tested two simple strategies: UNIFORM and ALLONONE. The UNIFORM strategy assigns an equal number of agents to all tasks that are not complete. If a task cost reaches zero, it is marked as complete and the agents are redistributed. The ALLONONE strategy assigns all agents to a single task until that task is completed. The order in which tasks are selected is from left to right in the "Initial Costs" column of Table 1. We experimented with different types of growth functions (Table 1). All experiments in the section use 20 agents with with work rate $w = 0.015$. The coefficients for w and in the growth functions are small to minimize the discretization of the time steps. A travel time of zero between all tasks is used to have a direct comparison as the optimal solution is only known for this case.

When there is no travel time, three different families of growth functions can be classified based on $\frac{\partial^2}{\partial x^2} h(x)$ as shown in [9]. If $\frac{\partial^2}{\partial x^2} h(x) < 0$, then the growth function is concave and the optimal solution is to have all agents on the task that is the smallest. In the case of a tie for the smallest, one task should be picked randomly and all agents should work on that one. When $\frac{\partial^2}{\partial x^2} h(x) = 0$, the growth is proportionally to the current cost of the task. In this linear case, any assignment that fully utilizes all agents at every time step is optimal. This is easily achieved since agents require no time to move between tasks. The last case is when $\frac{\partial^2}{\partial x^2} h(x) > 0$, which is when the growth is convex. This has an optimal solution of assigning agents to the largest task, and balancing the assignment equally among the largest tasks when ties exist. There are no known optimal solutions if tasks have different families of growth functions. For example if the first task grows convexly and the second task grows with a concave function, no known optimal solution exists even without travel time. Functions that have both concave and convex parts, such as the sigmoid function, also have unknown optimal solutions.

Table 1 compares the algorithms across seven different sets of growth functions, $h_i(x)$, which each have different optimal solutions. Each algorithm is largely deterministic, but we provided the average over five runs to ensure small tie breakers did not effect the results. The settings of the experiment, shown in Table 1, are: two convex,

one linear, two concave, one group of tasks with mixed functions and one function that has both concave and convex parts. The first convex function has two tasks, both with growth rate $h_i(x) = 0.000016 \cdot x^3$. The first task has an initial cost of 20, while the second task has an initial cost of 15. The second set of convex growth tasks all have a growth rate, $h_i(x) = 0.00019 \cdot x^2$ and initial costs $f_1^0 = 25, f_2^0 = 20, f_3^0 = 10$. For linear growth functions, $h_i(x) = 0.00360 \cdot x$ with initial costs $f_1^0 = 40, f_2^0 = 30, f_3^0 = 10$. The first concave tasks all have $h_i(x) = 0.020 \cdot \sqrt{x}$ and initial costs $f_1^0 = 20, f_2^0 = 15, f_3^0 = 10, f_3^0 = 5$. The second set of concave tasks grow very slowly, all with $h_i(x) = 0.02 \cdot \ln(x + 1)$ and initial costs $f_1^0 = 40$ and $f_2^0 = 30$. The $x + 1$ inside the logarithmic function is to ensure continuity of task growth when tasks near completion. Optimal solutions are not provided for the last two experiments, as there are no known solutions. The mixed category has one of each type of growth function: $h_1(x) = 0.00019 \cdot x^2$ and $f_1^0 = 25$, $h_2(x) = 0.00360 \cdot x$ and $f_2^0 = 10$ and $h_3(x) = h_i(x) = 0.020 \cdot \sqrt{x}$ with $f_3^0 = 10$. A growth rate corresponding to the sigmoid function is $h_1(x) = h_2(x) = 0.1 \cdot \frac{e^{-0.1 \cdot x}}{(1+e^{-0.1 \cdot x})^2}$ and has two tasks with initial costs $f_1^0 = 40$ and $f_2^0 = 30$. This is the only growth function which does not have a continuous first derivative when tasks are completed, namely $h(0) > 0$. However, when the cost of a task reaches zero at some time $\hat{t}$, i.e. $f_i^{\hat{t}} = 0$, we set $h(f_i^t) = 0$ for all $t \geq \hat{t}$.

We see that max-sum performs quite close to optimal in all cases when the optimal is known, and only loses to ALLONEONE on the sigmoid function. All strategies perform approximately the same for linear growth functions, as the theory suggests. The small differences are discretization effects of the agents. RT-LFF and UNIFORM only do well comparatively on the convex functions. The ALLONEONE performs well in general on the concave functions and the sigmoid. Next we will give a detailed analysis of the reasons for the algorithms performance.

The optimal solution for convex functions spends the majority of the time with an equal number of agents on all tasks. Max-sum follows very close to the optimal solution's assignment at every step, and thus performs similarly. RT-LFF attempts to minimize the number of agents switched and causes both tasks to finish at the same time. With $h_i(x) = 0.000016 \cdot x^3$, RT-LFF start with 15 agents on the first task, the larger, and 5 on the second. It then slowly switches agents from the larger task to smaller task, until about halfway through the simulation. At this point, both tasks have the same cost and agent are split 10 on each for the rest of the simulation. UNIFORM also performs well for convex functions for the same reason. ALLONEONE has a strategy very different than the optimal, so it is not surprising that it does poorly for this group of functions.

The optimal strategy for concave growth functions is to have all the agents on the smallest task. Again, max-sum finds this solution and performs competitively with the optimal solution. RT-LFF and UNIFORM perform poorly on this family of functions, as they both spread agents over all tasks. Although ALLONEONE is similar to the optimal solution, it assigns all agents to the larger tasks first instead of the smaller ones. This causes ALLONEONE to perform sub-optimally, but not as poorly as the drastically different strategies of RT-LFF and UNIFORM.

The optimal solution is not known for mixed functions. Max-sum outperforms all others in this case, but the assignments vary drastically over time as shown in Figure 2. ALLONEONE attempts to finish the first task, which grows considerably more than the

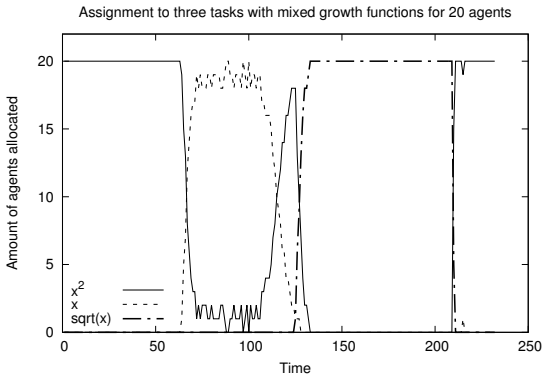


Fig. 2: Agent assignment across all three tasks with different growth types.

others. RT-LFF also assigns most of the agents to this first task for the majority of the simulation, causing RT-LFF and ALLONONE to perform similarly. UNIFORM does not assign enough agents to the important first task, initially 7, which is not enough to reduce the task growth. Eventually when the second task finishes about 25% through the simulation, the first task gets 10 agents which are able to overcome the task growth.

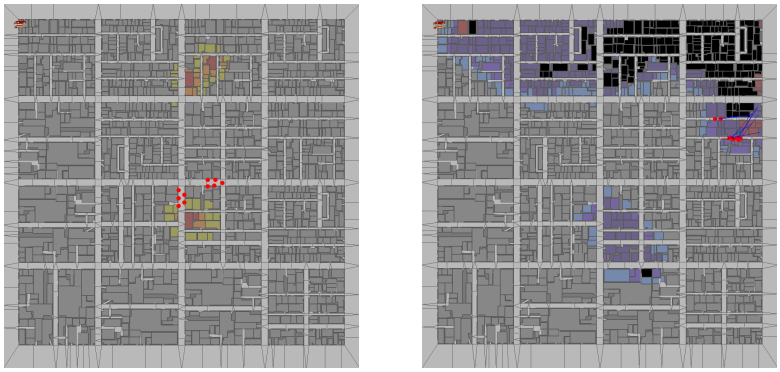
The sigmoid function has a steep slope when the cost is small. ALLONONE is able to overcome this fairly easily as all agents are on one task. Max-sum initially distributes agents across both tasks, but then moves them to a single task when the costs are smaller. Both RT-LFF and UNIFORM attempt to complete both tasks at the same time for the whole simulation, thus more effort to get past the steep slope when the cost is low.

Using ANOVA with repeated measures gives a p-value of 0.0716, which is close to being statistically significant so we did paired t-tests. The most different pair was max-sum and UNIFORM, with a p-value of 0.015. All other pairs did not reach statistical significance, including compared to the optimal when it was known. As this experiment spans different growth rates, no single algorithm could consistently be the best.

5.2 RoboCup Rescue Simulator

In this section we focus on fires in the RoboCup Rescue simulator. The RoboCup Rescue simulator is designed for urban search and rescue after an earthquake, where buildings collapse and fires start in buildings. The environment is complex with thousands of buildings and hundreds of agents in the full simulator which uses street maps of real cities. The full simulator uses heterogeneous agents, but for this work we focus only on the agents that can extinguish fires, i.e. firetrucks, through the use of the RMASBench simulator extension [4]. Although RMASBench is an extension of the RoboCup Rescue Agent Simulator, the communication is relaxed to allow many more messages to be sent than in the original simulator, along with multiple messages per time step.

Fires are the most dangerous hazard in RoboCup Rescue. Buildings heat up and catch on fire based on how many other nearby buildings are on fire. This creates a positive feedback loop, which causes fires to grow at exponential rates. A screenshot of



(a) Starting state of VC 3 with 95% of buildings intact.

(b) Max-sum at end with 73% of buildings intact.

Fig. 3: Virtual City 4 (VC4)

the simulator is shown in Figure 3a and Figure 3b. Buildings on fire are yellow, orange and red in increasing intensity and temperature. When a fire is extinguished, buildings become blue or purple. When a building is destroyed it turns black.

We cluster nearby fires into a single task, where the cost is the number of buildings on fire. We use bottom-up hierarchical clustering with the Euclidean distance as the metric and the minimum distance between all pairs linkage criterion. Clustering ends when the distance between the closest pair of clusters is over 50 meters.

Both $h(x)$ and w need to be estimated. We approximate the cost growth function for all tasks as $h(f_i^t) = g \cdot f_i^t - |n_i^t| \cdot w$ as in Eq. 3 to Eq. 5. To estimate g, buildings were allowed to burn unhindered for 100 simulation steps in 20 different tests. An exponential regression fit best, so we could model it as $a \cdot e^{b \cdot t}$ or simply $e^{b \cdot t}$, where b is the g we want. There are two regression models that minimize the sum of squares: $4.6852 \cdot e^{0.0393 \cdot t}$ or $e^{0.0561 \cdot t}$. We cannot use the first regression model since the scaling factor outside the exponent is reserved for the initial task cost. Thus we have to decide between the two g values, 0.0393 and 0.0561. We do it by looking at how they affect the distribution of w.

The work rate, w, was also empirically derived. A fixed small number of fires were repeatedly extinguished in 70 tests. If f_i^0, g, $|n_i^t|$ and t_c are known, Eq. 4 can then be used to solve for w. The experiments found that w is rather noisy and typically has a few high outliers. This means in some cases, the fire trucks were able to extinguish the fire much more rapidly than the model anticipated. The lack of low outliers provides stability, since this indicates agents rarely extinguish poorly. We chose $g = 0.0393$, due to a smaller variance in w and fewer outliers.

This exponential model corresponds to the linear task growth from Section 5.1. In the simple simulator all strategies performed fairly identical in this case, but our model for RoboCup is an approximation of the real process. Max-sum is overly responsive to changes, so when a few buildings unexpectedly start on fire or get extinguished, agents are transferred. This can cause assignment thrashing for max-sum, as agents in transit are lost work and the solution quality decreases overall. RT-LFF is much more

Table 2: Percent of buildings damaged at the end of simulation in RoboCup.

Map	RT-LFF		Max-Sum	
	μ	σ	μ	σ
VC 1	**72.14**	**7.23**	83.82	6.03
VC 2	**1.96**	**0.43**	21.02	0.33
VC 3	50.41	9.70	**26.19**	**6.36**
VC 4	**29.02**	**2.90**	56.53	4.50
VC 5	**76.90**	**2.69**	82.42	1.34
Paris 1	**15.47**	**4.55**	27.46	5.03
Paris 2	**58.89**	**7.00**	68.63	8.73
Paris 3	39.27	4.69	**31.19**	**3.92**
Paris 4	**8.76**	**0.45**	9.01	0.52
Paris 5	25.22	6.36	**20.36**	**5.66**

conservative and rarely thrashes, however it is unable to exploit smaller gains. This causes both methods to perform similarly in this domain on average.

Table 2 shows the percentage of damaged buildings for 5 variations of both the Virtual City (VC) and Paris map. Each variation is denoted by a number after the map name. Changes are to the number and location of fires along with the number and location of agents. Virtual City is a smaller map, so the number of agents and fires range between 8 to 15 and 30 to 50 respectively. In Paris there are between 10 to 50 agents with 50 to 120 fires. This randomization causes some configurations to be easier than others. Each configuration was run 5 times. Both RT-LFF and max-sum use the clustered building data as an approximation of f_i^t. Results on maps such as VC 5 are poor across all algorithms. This indicates more the hardness of the VC 5 configuration than the inefficiency of the algorithms. The opposite is the case in Paris 4, where the configuration is too easy and all algorithms performed well.

Often the initial configuration of a map makes it either too hard or too easy, which causes results to have bimodal distributions [10]. In this case the scores are similar across algorithms making differentiation more difficult. Due to the bimodal nature we apply the non-parametric Wilcoxon signed-rank test instead of the normal t-test. Neither method showed a statistically significant advantage over the other.

6 Conclusions and Future Work

In this work we focused on tasks with costs that grow over time and provided the first decentralized task assignment approach for this type of tasks. Our decentralized solution, which is based on the max-sum algorithm, performs similarly to the previous centralized solution in both a simple simulation and the RoboCup Rescue Agent Simulator. Our functions that predict the future cost of tasks assume that the number of agents active on a task is constant. However, sometimes it is beneficial to reassign agents before the task is completed. Future work will consider the effect of planned reassignments over time. The current work also assumes that these functions are estimated upfront, but they could be learned over the course of the simulation. Given different families of

functions, the model could be refined over time to select the best function family with the best parameters. This could also help determine what errors come from modeling versus inherent randomness of the growth.

Acknowledgments: Work supported in part by NSF-IIP-1439728 and the Graduate School of the University of Minnesota.

References

1. S. Amador, S. Okamoto, and R. Zivan. Dynamic multi-agent task allocation with spatial and temporal constraints. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1384–1390, 2014.
2. A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 639–646, May 2008.
3. H. Kitano and S. Tadokoro. RoboCup Rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine*, 22(1):39–52, 2001.
4. A. Kleiner, A. Farinelli, S. Ramchurn, B. Shi, F. Maffioletti, and R. Reffato. RMASBench: Benchmarking dynamic multi-agent coordination in urban search and rescue. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1195–1196, 2013.
5. F. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, 47(2):498–519, Feb 2001.
6. J. Melvin, P. Keskinocak, S. Koenig, C. Tovey, and B. Ozkaya. Multi-robot routing with rewards and disjoint time windows. In *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 2332–2337, Oct 2007.
7. P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2004.
8. J. Parker and M. Gini. Tasks with cost growing over time and agent reallocation delays. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 381–388, 2014.
9. J. Parker and M. Gini. Controlling growing tasks with heterogeneous agents. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, 2016.
10. J. Parker, J. Godoy, W. Groves, and M. Gini. Issues with methods for scoring competitors in RoboCup Rescue. In *Autonomous Robots and Multirobot Systems at AAMAS*, 2014.
11. T. Penya-Alba, M. Vinyals, J. Cerquides, and J. A. Rodriguez-Aguilar. A scalable message-passing algorithm for supply chain formation. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1436–1442, 2012.
12. M. Pujol-Gonzalez, J. Cerquides, A. Farinelli, P. Meseguer, and J. A. Rodriguez-Aguilar. Efficient inter-team task allocation in RoboCup Rescue. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 413–422, 2015.
13. M. Pujol-Gonzalez, J. Cerquides, P. Meseguer, J. A. Rodríguez-Aguilar, and M. Tambe. Engineering the decentralized coordination of UAVs with limited communication range. In *Advances in Artificial Intelligence*, pages 199–208. Springer, 2013.
14. S. Ramchurn, A. Farinelli, K. Macarthur, M. Polukarov, and N. Jennings. Decentralised coordination in RoboCup Rescue. *The Computer Journal*, 53(9):1–15, 2010.
15. D. Tarlow, I. E. Givoni, and R. S. Zemel. Hop-map: Efficient message passing with high order potentials. In *Proc. Int'l Conf. on Artificial Intelligence and Statistics*, pages 812–819, 2010.
16. X. Zheng and S. Koenig. Reaction functions for task allocation to cooperative agents. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 559–566, 2008.
17. R. Zivan. Anytime local search for distributed constraint optimization. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1449–1452, 2008.